Database Creation - Handling

CREATE

```
#creating a new table
CREATE TABLE TableName (
        ColumnName1 DataType,
        ColumnName2 DataType,
        ColumnName3 DataType ) ;

#creating table with NOT NULL constraints
CREATE TABLE TableName (
        ColumnName1 DataType NOT NULL,
        ColumnName2 DataType NOT NULL,
        ColumnName3 DataType ) ;
```

DEFAULT

```
#setting a default value for a column value
CREATE TABLE TableName (
        ColumnName1 DataType NOT NULL,
        ColumnName2 DataType DEFAULT ‘San Jose’,
        ColumnName3 DataType ) ;
```

UNIQUE

```
#setting a unique constraint on the column value - NULL is exempted
CREATE TABLE TableName (
        ColumnName1 DataType NOT NULL,
        ColumnName2 DataType UNIQUE,
        ColumnName3 DataType ) ;
```

ALTER

```
#changing the schema of the table - adding a new column
ALTER TABLE TableName
ADD newColumnName DataType ;

#adding a  new column with a default value
ALTER TABLE TableName
ADD newColumnName DataType DEFAULT ‘San Jose’ ;
```

PRIMARY KEY

```
#one label for each row
CREATE TABLE TableName (
        id INTEGER PRIMARY KEY,
        ColumnName2 DataType NOT NULL,
        ColumnName3 DataType ) ;
```

DROP

```
#deleting a table from database
DROP TABLE TableName;

#Checking if the table exists before deleting
DROP TABLE IF EXISTS TableName ;
```

Queries

SELECT

```
#most simplest ever data source
SELECT ‘Hello World’ ;

#selecting all rows & columns
SELECT * FROM TableName;

SELECT ColumnName1, ColumnName2 FROM TableName;

#creates an alias name for table name
SELECT * FROM TableName AS t;

#creating an alias name for column name
SELECT ColumnName1 As “Result Column” FROM TableName;
```

ORDER BY

```
#sorting results - default ascending
SELECT * FROM TableName ORDER BY ColumnName1;

#sorting results in ascending
SELECT * FROM TableName ORDER BY ColumnName1 ASC;

#sorting results in descending
```

```
SELECT * FROM TableName ORDER BY ColumnName1 DESC;

#sorting results based on multiple columns
SELECT * FROM TableName ORDER BY ColumnName1, ColumnName2;

#constraints on rows returned to 10 (or N)
SELECT * FROM TableName ORDER BY ColumnName1 LIMIT 10;

#selecting the next 10 rows of the previous query
SELECT * FROM TableName ORDER BY ColumnName1 LIMIT 10 OFFSET 10;

#checking for missing values
SELECT * FROM TableName WHERE ColumnName1 IS NULL
```

GROUP BY

```
#aggregation inside a table -prints counts of rows for each distinct value in col 1
SELECT ColumnName1, COUNT(*) FROM TableName GROUP BY ColumnName1 ;

#using joins along with group by
SELECT table1.table1Column , COUNT(table2.table2Column)
        FROM TableName1 as table1
        JOIN TableName2 as table2
                ON table1.id = table2.id
        GROUP BY table1.id
        ORDER BY table2.table2Column ;
```

HAVING

```
#similar to a where clause but used for Group by
SELECT ColumnName1 , ColumnName2 FROM TableName
        WHERE ColumnName3 = 'San Jose'
        GROUP BY ColumnName1
        HAVING ColumnName2 > 200 ;
```

SELECT with WHERE

```
#single condition with string check
SELECT * FROM TableName WHERE ColumnName1 = 'JACKKNIFE';

#order by with WHERE clause
SELECT * FROM TableName WHERE ColumnName1 = 'Jetty' ORDER BY ColumnName2
DESC;

#setting alias when selecting using conditions
```

SELECT ColumnName1 as BetterColumnName FROM TableName WHERE ColumnName2 = ‘San Jose’;

COUNT

#counts number of rows
SELECT COUNT(*) FROM TableName;

#Counts rows based on conditions
SELECT COUNT(*) FROM TableName WHERE ColumnName1 = ‘San Jose’;

#Counts specific column rows (ie) without Null values
SELECT COUNT(ColumnName1) FROM TableName WHERE ColumnName2 = ‘San Jose’;

OR

#implements OR condition on WHERE clause
SELECT * FROM TableName WHERE ColumnName2 = ‘San Jose’ OR ColumnName2 IS NULL ;

AND

#implements AND condition on where clause
SELECT * FROM TableName WHERE ColumnName2 = ‘San Jose’ AND ColumnName2 IS NOT NULL ;

LIKE

Finds pattern in ColumnName2- - this case, any word with ‘san’ in it
SELECT * FROM TableName WHERE ColumnName2 LIKE ‘%san%‘ ;

Finds in ColumnName2- - any word ending with ‘san’
SELECT * FROM TableName WHERE ColumnName2 LIKE ‘%san‘ ;

Finds in ColumnName2- - any word starting with ‘san’
SELECT * FROM TableName WHERE ColumnName2 LIKE ‘san%‘ ;

Finds in ColumnName2- matching a single character ‘a’, in the second position
SELECT * FROM TableName WHERE ColumnName2 LIKE ‘_a%‘ ;

IN

#selecting values that belongs in a list
SELECT * FROM TableName WHERE ColumnName1 IN (‘San Jose’ , ‘Menlo Park’ , ‘Santa Clara’) ;

CASE WHEN.. THEN.. ELSE.. END

```
#implementing if conditions in SQL - here, checking if NULL or not
SELECT
        CASE WHEN ColumnName1 THEN ‘True’ ELSE ‘False’ END AS Column1Bool ,
        CASE WHEN ColumnName2 THEN ‘True’ ELSE ‘False’ END AS Column2Bool
FROM TableName ;

#implementing if conditions in SQL - here, checking against a value
SELECT
        CASE ColumnName1 WHEN 1 THEN ‘True’ ELSE ‘False’ END AS Column1Bool ,
        CASE ColumnName2 WHEN 0 THEN ‘False’ ELSE ‘True’ END AS Column2Bool
FROM TableName ;
```

INSERT

```
#Specifying all column values, the left out table cols with be set to NULL
INSERT INTO TableName  (ColumnName1 , ColumnName2, ColumnName5 )
 VALUES (ValueForCol1 , ValueForCol2, ValueForCol5 );

#inserting default values
INSERT INTO TableName DEFAULT VALUES;

#inserting rows from selected rows from another table
INSERT INTO TableName1 (ColumnName1, ColumnName2 )
SELECT Column1 , Column2 FROM TableName2 WHERE Column2 = ‘Midnight’ ;
```

JOIN

Documentation

```
#Left join
SELECT left.ColumnNameLeft , right.ColumnNameRight
FROM LeftTableName as left
JOIN RightTableName as right ON right.id = left.id ;

#junctioning multiple tables
SELECT table1.Column1, table2.Column2,  table3.Column3
FROM Table3Name as table3
JOIN Table1Name as table1 ON table1.id = table3.id
JOIN Table2NAme as table2 ON table2.id = table3.id ;

#left outer join
SELECT table1.Column1, table2.Column2,  table3.Column3
FROM Table3Name as table3
```

```
LEFT JOIN Table1Name as table1 ON table1.id = table3.id
LEFT JOIN Table2NAme as table2 ON table2.id = table3.id ;
```

UPDATE .. SET

```
#updating column values
UPDATE TableName
SET ColumnName1 = 'Bay Area', ColumnName2 = '95100'
WHERE ColumnName1 = 'San Jose' ;
```

DISTINCT

```
#selecting distinct values in column
SELECT DISTINCT ColumnName1 FROM TableName;
```

DELETE

```
#deleting a specific row
DELETE FROM TableName WHERE ColumnName1 = 'Los Angeles' ;
```

Data types

TYPEOF

```
#finding the type of something
SELECT TYPEOF (IntegerColumnName + 2) ;
```

CAST

```
#converting data types- this ex. will return 0.5
SELECT CAST(1 as REAL) / 2 ;
```

ROUND

```
#rounding of floating points
SELECT ROUND(2.333333) ; #returns 3
SELECT ROUND(2.55555) ; #returns 2.556
```

NULL

```
#testing for NULL values
SELECT * FROM TableName WHERE ColumnName1 IS NULL;

#testing for NOT NULL values
SELECT * FROM TableName WHERE COlumnName1 IS NOT NULL;
```

LITERAL STRING

```
SELECT ‘Literal String’ ;

#using a single quote in a string
SELECT ‘Literal’’s String’ ;

#concatenating strings - Postgres
SELECT ‘This’ || ‘ is ‘ || ‘ very ‘ || ‘awkward’ ;

#concatenating strings - MySQL
SELECT CONCAT (‘This’, ‘ is ‘, ‘ better ‘) ;
```

LENGTH

```
#SQLite - Finding length of string
SELECT LENGTH(‘This is my String’) ;
SELECT ColumnName1, LENGTH(ColumnName1) as length FROM TableName
        ORDER BY length DESC ;
```

SUBSTR

```
#SQL
Lite - finding a substring - this ex. prints the word ‘is’
SELECT SUBSTR(‘This is also my string’ , 6 );

#SQLLite - finding a substring - this prints date, month, year separately: dd/mm/yyyy
SELECT DateColumn,
        SUBSTR (DateColumn,1,2) AS Date,
        SUBSTR (DateColumn,4,2) AS Month,
        SUBSTR (DateColumn,7,4) AS Year,
FROM TableName ;
```

TRIM

```
#crops a string off spaces
SELECT TRIM (‘  String ‘) ;
SELECT TRIM (‘  String String  ‘) ; #cuts off only from both ends
SELECT LTRIM (‘  String ‘) ;  #cuts off left side
SELECT RTRIM (‘  String ‘) ;  #cuts off right side
SELECT TRIM (‘\\\\\\String\\\‘ , ‘\’) ;  #cuts off \
```

UPPER

```
#changing string to upper case - this ex. will return TRUE
SELECT UPPER(‘string’) = ‘STRING’ ;
```

LOWER

```
#Changes string to lower case - this will return TRUE
SELECT LOWER('STRING') = 'string' ;
```

NUMBERS

```
INTEGER (precision)
DECIMAL(precision, scale)
MONEY (precision, scale)
REAL (precision)
FLOAT (precision)
```

Dates and Times (sqlLite)

```
# printing current date time
SELECT DATETIME("now")

#printing current date
SELECT DATE("now")

#printing current time
SELECT TIME("now")

#printing tomorrow's datetime
SELECT DATETIME ("now", '+1 day') ;
```

Transactions

```
# kinda like a function for faster performance -MySQL
START TRANSACTION:
        INSERT INTO TableName (Column1, Column2 ) VALUES ( 2 , 'Bogus' ) ;
        UPDATE TableName SET Column1 = (Column1 +1 ) WHERE Column2 =  'Bogus' ;
END TRANSACTION ;

#SQLlite
BEGIN TRANSACTION:
        INSERT INTO TableName (Column1, Column2 ) VALUES ( 2 , 'Bogus' ) ;
        UPDATE TableName SET Column1 = (Column1 +1 ) WHERE Column2 =  'Bogus' ;
END TRANSACTION ;

# MySQL rolling back -  doesn't update tab;e
START TRANSACTION:
        INSERT INTO TableName (Column1, Column2 ) VALUES ( 2 , 'Bogus' ) ;
```

```
	UPDATE TableName SET Column1 = (Column1 +1 ) WHERE Column2 =  'Bogus' ;
ROLLBACK ;
```

Triggers

```
#useful when u want to create data logging
#creating triggers - helpful to make changes on different tables depending on one table
CREATE TRIGGER triggerName AFTER INSERT ON TableName
	BEGIN
		#any query
		UPDATE TableName SET ColumnName1 = NEW.id
			WHERE ColumnName2 = NEW.ColumnName2
	END ;

#preventing changes using triggers
CREATE TRIGGER triggerName BEFORE UPDATE ON  TableName
	BEGIN
		SELECT RAISE(ROLLBACK, 'Cannot perform task, sowie!' )
		FROM TableName
		WHERE ColumnName1 = 'True' ;
	END ;
```

Subselect

```
#better organising - example 1
SELECT table1.table1Col , table2.table2Col
	FROM ( SELECT ColumnName1, ColumnName2 FROM TableName2 ) AS table2
	JOIN TableName1 AS table1 ON table1.table1idColumn =
	table2.table2referenceColumn ;

#example 2
SELECT ColumnName1, ColumnName2
	FROM TableName1
	WHERE ColumnName1 IN ( SELECT NameColumn FROM TableName2) ;

#example 3
SELECT table1.ColumnName1,  table1.ColumnName2
	FROM TableName1 AS table1
	JOIN (SELECT table2Col1, table2Col2 FROM TableName2
		WHERE table2Col2 > 20 ) AS table2
	ON table1.id = table2.id ;
```

Views

CREATE VIEW

#using subselects over and over again - create and run view once and use it!
CREATE VIEW ViewName AS SELECT ColumnName1, ColumnName2 FROM TableName
;

#will get saved as ViewName, then..
SELECT * FROM ViewName ;

DROP VIEW

#deletes views
DROP VIEW IF EXISTS ViewName ;

Calculations

Sum of a column

SELECT SUM(ColumnName) FROM TableName

Minimum Value in a column

SELECT MIN(ColumnName) FROM TableName

Maximum Value in a column

SELECT MAX(ColumnName) FROM TableName

Average value of a column

SELECT AVG(ColumnName) FROM TableName

Standard Deviation Check to find Outliers

1) Calculate Mean of the column
2) Calculate Std Deviation of the column
3) Set an upper threshold equals to Mean + (3 * Std Dev)

4) Set an lower threshold equals to Mean - (3 * Std Dev)
5) Any values lesser or greater than these values are OUTLIERS